Chrome Extension Policy Analysis

By Bamlak, Isak, Robert, Yaateh

1. Abstract

With over a 63% market share of internet browsers, Google Chrome is by far the most popular way to browse the internet [1, 7, 8]. One feature that lends to its popularity is Chrome's rich library of extensions with over 180,000 different extensions available for download [9]: after paying a $5 license fee a user can freely upload extensions to be almost immediately available for download. That presents a possible attack vector for malicious agents to target users.

In this paper we investigated multiple exploits that could be used by malicious agents to collect user information via extensions and how extensions could be abused. We collected 2,750 extensions from the chrome web store and found that a large portion of them ask for powerful permissions and follow dangerous code practices. In particular, over a third of the extensions downloaded do not follow at least one security guideline set forth by Google. We provide insight into what Google currently has done to prevent these kinds of exploits, suggest simple tweaks that could be made to the current extensions submissions process that would help developers make their extensions safer, and changes that would provide less tech savvy users with the context needed to understand the risk of downloading a specific extension.

2. Introduction

The most widely used web browser today is Google Chrome [1]. Consequently, that makes Chrome the most prone to cyberattacks from malicious actors. While there are many sections of chrome that can be attacked, this project focuses on chrome extensions and the policies around it.

In deciding how to approach this project, we asked the following questions: how easy is it for a malicious developer to publish malicious chrome extensions and how easy is it to make a Chrome extension malicious? Chrome extensions expose the threat that because they are not made by Google and because of the sheer scale of the extension ecosystem it's not feasible to manually evaluate every version of a published extension for malicious activity. In fact, when initially signing an agreement to become a developer to publish applications, the agreement says, "while Google is not obligated to monitor the Products or their content, Google may at any time review or test your Products and their source code for compliance with this Agreement" [3]. Instead they've developed a secret security verification process, the details of which are not publicly available [6].

Our process involved figuring out the ways that extensions can be or become malicious. What we mean by malicious is the following, perhaps relaxed, three definitions: (1) an extension is malicious if it can access user information that it's not explicitly permitted to access without

the user knowing, (2) an extension is malicious if it can perform actions on a user's browser that it's not explicitly permitted to perform without the user knowing, or finally, (3) an extension is malicious if it does not comply with any of the security policies set forth and recommended by Google Chrome. Thus, we explored various loopholes that allowed Chrome extensions to be malicious. In particular, we researched previous attempts and documented findings on malicious extensions [2], we analyzed Chrome's policies around publishing extensions, and we analyzed a collection of 2,750 chrome extensions for compliance with policies and recommendations from the Chrome developer website. The general findings suggest that Chrome tries its best to serve both its users and its developers but implicitly places the burden on users to know what not to download and flag extensions as malicious or inappropriate—that is, if any extension violates the agreement for being allowed to be published in the Chrome Web Store.

Overall, this is important because the scale of Chrome puts several users' private information and online-security on the line. In thinking about the average Chrome user, it might not be safe to assume that they are tech-savvy enough to know whether an extension is malicious, and often even tech-savvy users don't investigate how secure the chrome extensions are. In fact, Chrome does not reveal a lot of details about its automated review process. So, in this paper, we hope to bring more attention to what can go wrong and nudge Chrome to improve on its communication to users about the various processes involving Chrome extensions.

3. Background

Below we explore other papers that have studied chrome extensions in the past and bring up what kind of resolutions they found.

Security Analysis of Chrome Extensions [2016]

This group found methods to exploit Chrome permissions through those Google deemed as not requiring any special permissions. Presumably, a user could utilize any of these with no worry of attacks or exploits. The group was able to identify 7 "non-special" permissions with vulnerabilities to the user: *chrome.tabs, chrome.system.cpu, chrome.system.display, chrome.system.storage, chrome.sessions, Chrome.systems, memory, window*.

Through these permissions, a malicious agent could gain useful CPU data, display information, etc. Along with these data vulnerabilities, there was also potential to carry out several attack vectors:

- ***chrome.power*:** draining a users power more rapidly
- ***chrome.notifications*:** presenting a user with persistent notifications
- ***chrome.clipboardWrite*:** writing any desired text to a users clipboard
- ***chrome.fontSettings*:** making pages unreadable and difficult to close, which can be a Denial-of-service attack

- **Denial-of-service (DoS) attack:** closing all tabs and making it impossible to open any Chrome windows
- **Phishing:** using the *chrome.tabs* permission, redirecting users to undesired URLs and subsequently taking information with a covert alias.

As such, extensions provide malicious agents with another method to attack a user, one that many users might implicitly trust because it appears to have Google's seal of approval of not using noteworthy permissions.

Exploiting Leaky Chrome Extension API [2019]

This group also identified some security vulnerabilities. They were able to find that extensions could bypass the Same Origin Policy, which is put in place to block malicious websites and scripts, by using cross-origin permissions. In this case, a malicious website could bypass the policy and gain read and write access to any vulnerable data without the user's knowledge. The group was able to design a threat model, wherein malicious websites could access user cookies through any extension that enables cookie permissions. This attack could extend to accessing a user's entire browsing history (through chrome.history) and manipulating downloads to place files on a user's device (through chrome.downloads). Clearly, a number of exploits could be carried out because of leaky extensions, and the group identified a need for stronger requirements for deployment, as these weak extensions may act as a middleman for malicious behavior from websites.[14]

Updates on Security Policy

In October 2018, Google launched Project Strobe in response to severe data exposures found within the Google+ product. [15] The project hopes to make several changes to the developer requirements:

- Extensions that handled sensitive and personal user data (as defined by the Google security policy) were mandated to articulate their policies, but now any applications or extensions that facilitate personal communication and user-generated content must also state their policies.
- Access to Gmail data is significantly tightened and 3rd party access to Drive is limited.
- New privacy policy requirements force developers to explain exactly how data is used to ensure minimal necessary data access.
- More rigorous extension review process will reject anything that does not adhere to the above.

These updates certainly look promising, as e. Many still question the effectiveness of these changes, however, as the extent of these requirements' success is entirely dependent on the rigor of this new-and-improved review process.

With respect to the attack vectors the group determined, some vulnerabilities look to still be present. DoS attacks have not been addressed and phishing remains a large and difficult issue. However, in April 2020, Chrome again updated the security policies for the Chrome Web Store to plug some of the issues. [16] The new additions combat:

- Multiple extensions that serve the same function
- Misleading descriptions, names, icons, etc.
- Extensions that launch other apps or websites without user consent
- Extensions that spam notifications and advertisements
- Extensions that send messages before the recipient's confirmation

Though Chrome has taken many steps to mitigate the known vulnerabilities within the extension architecture, many exposures still exist, and the road to perfect security is a long one.

Content Security Policies (CSPs)

One of the core tenets of internet security is Content Security Policy (CSP). They key idea is that a large part of risk while browsing the internet is a malicious party loading code to be executed on your computer. In order to combat this, CPSs establish sets of rules as to where any resource can be fetched from. For example a particular extension might only allow resources to be fetched from *https://www.imgur.com* and the current tabs domain. This prevents a malicious script from another source to be injected and executed. CSPs can go even further, banning inline scripts entirely and forcing all scripts to be stored on trusted sources. A good extension will keep as strict a CSP as possible, thus limiting malicious parties' attacks surface to the bare minimum. More information about CPSs can be found on the World Wide Web Consortium's Website [12].

To that end, Chrome has added support for CSPs into extensions in the second manifest version [10] and has completely phased out the use of the first manifest version since January 2014, meaning that no chrome extension with the first manifest version will be allowed to be submitted to the Chrome Web Store nor will be run in any user's chrome browser [11]. We will discuss how adding CSPs to chrome extensions provide much more security and how they still can be better enforced.

4. Methods

4.1 Current Chrome Extension Publishing Policies

The process through which a developer goes from creating an extension to publishing relies on multiple checkpoints set by Google to ensure that a given application (which may be an extension) published does not violate multiple terms set forth by Google. In particular, a user must agree with the Google Chrome Web Store Developer Agreement, the Google Privacy Policy,

and the Chrome Webstore Developer Program Policies [3, 4, 5], and in addition they must pay a $5 registration fee (see Figure 5.1).

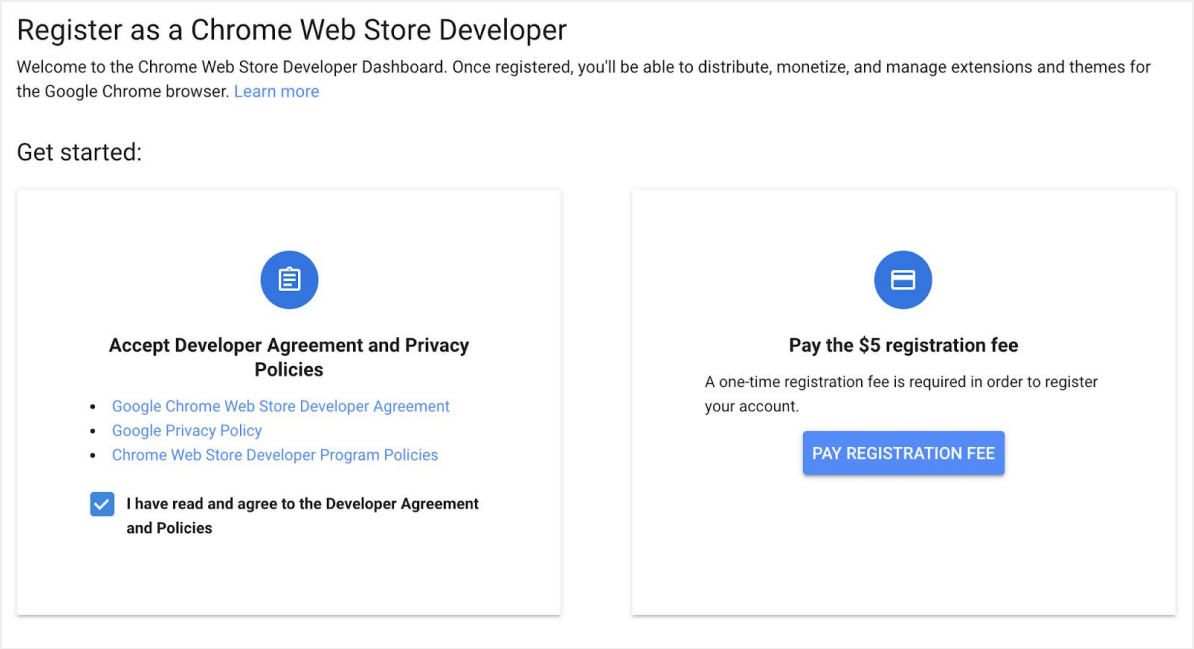


Figure 4.1: Agreements and Payment to be allowed to publish in the Chrome Web Store as a developer.

More specifically, the Chrome Webstore Developer Program Policies disallow all the following types of content: Sexually Explicit Material, Violent or Bullying Behavior, Hate Speech, Impersonation or Deceptive Behavior, Intellectual Property, Illegal Activities, Malicious Products, Code Readability Requirements (while code minification is allowed, "Developers must not obfuscate code or conceal functionality of their extension."), and Prohibited Products. This is a long list that developers must agree to. In particular, the Malicious Product [5] sections says:

> *Don't transmit viruses, worms, defects, Trojan horses, malware, or any other items of a destructive nature. We don't allow content that harms or interferes with the operation of the networks, servers, or other infrastructure of Google or any third-parties. Spyware, malicious scripts, and password phishing scams are also prohibited in the Chrome Web Store.*

In addition, the Google Chrome Web Store Developer Agreement [3] says the following about violating any of the policies agreed to:

> *7.2. Google Review and Takedowns. While Google is not obligated to monitor the Products or their content, Google may at any time review or test your Products and their source code for compliance with this Agreement, the Google Chrome Web Store Program Policies, and any other applicable terms, obligations, laws, or regulations, and may use automated means to conduct such review. … If Google is notified by you or otherwise becomes aware and determines in its sole discretion that a Product or any portion thereof or your Brand Features; (a) violates the intellectual property rights or any other rights of any third party; (b) violates any applicable law or is subject to an injunction; (c) is pornographic, obscene or otherwise violates Google's hosting policies or other terms of service as may be updated by Google from time to time in its sole discretion; (d) is being published or distributed by you improperly; (e) may create liability for Google or any third party; (f) is deemed by Google to have a virus or is deemed to be malware, spyware or have an adverse impact on Google's or a third party's network; (g) violates the terms of this Agreement or the Google Chrome Web Store Program Policies; or (h) the display of the Product is impacting the integrity of Google servers (i.e., users are unable to access such content or otherwise experience difficulty), Google may prevent the Product from being made available in the Web Store, remove the Product from the Web Store, remotely disable or remove the Product from user systems or devices, or flag, filter, modify related materials (including but not limited to descriptions, screenshots, or metadata), or reclassify the Product at its sole discretion.*

So, while Google may not always run checks to verify that a given extension is compliant with its policies, it may periodically check whether the extension is compliant upon internal procedures or through a user complaint. The consequences for violating such policies, at the very least, is the removal of the developer's application and the risk of being banned from publishing any further application. In other places, the Chrome Webstore Developer Program Policies says that Google may report the appropriate authorities in case of a serious violation (for instance child pornography) [5].

This model has a few benefits, but it needs strengthening. First, this model allows good actors to be let into the developer platform and publish their extensions easily. They only have to pay a $5 registration fee. Secondly and more importantly, it discourages malicious actors from attempting to exploit the system, at least in obvious ways. In an ideal world, malicious extensions will be caught quickly through internal procedures—which we speculate to operate in a form of automated testing for obvious violations and periodic more extensive checks. Malicious actors would not benefit from any blatant transgressions, as the extensions will be quickly removed. In addition, if they get banned they must create a new Google account and register again—i.e., pay the $5 fee again. This could easily result in financial loss. At the same time, Chrome's user base is growing, the number of published extensions is increasing, and attackers are getting more sophisticated. Targeting even a small percentage of those users has huge potential for malicious actors, and as our analysis shows below, over a third of the extensions we surveyed are not complaints with Google's Developer Guidelines on CSPs in obvious ways, putting users at risk. Chrome needs to further strengthen its policies. We will

discuss possible modifications to their webstore in order to offer uneducated users a clear picture of what risks a particular extension could present.

4.2 Extension Architecture

In order to provide developers with the tools that extend Chrome Browser functionality without compromising integrity, Google has developed a very straightforward extension architecture. Its main emphases are (1) minimizing interaction between different code running in the browser, and (2) restricting access to potentially sensitive material through auditable APIs. Chrome extensions are mainly composed of three different sets of files: an extension's *manifest.json* file, several content script files, a *background.js* file accompanied with all the associated resource files such as icons and html pages.

manifest.json

The extension manifest can be considered the license and registration of the extension, it identifies basic information, what scripts can interact with the extension, and contains information regarding the permissions the extension has access to. Every Chrome extension must have a manifest, and as of January 2014, this manifest file must be of version 2 [11]. The various chrome APIs require their corresponding permission to be specified for use within the extension. Another important role of the extension manifest is to specify on what webpages the extensions code should be run. Specifically, when a web address specified in the manifest is accessed, the extension's *content-scripts* are called. Finally, the manifest specifies what sources other libraries and assets can be retrieved from.

Content

The content file is usually the meatiest part of an extension, it's the code that interacts directly with a webpage and its data. The content file is usually what contains most of the scripts in an extension and it's code is run in the context of the webpage from where the browser is calling it. As such the code in the content file has access to anything that may be open in that browser tab but is also limited to just that. This is meant to limit opportunities to leverage an extension for access to user information. If an extension is to do more than just be called on a specified set of web addresses then it needs to have a *background.js* file.

background.js

The *background.js* file is the code that runs whenever chrome is started. This allows for an extension to be listening for a specific event, say a specific button press or similar action. Whatever code is here is run separate from the actual tabs open and as such has no direct interaction with any open tabs. However, the code here can communicate with the code being run in content files. Information can be passed from a content file process to another via the background file code despite the background code being run separately.

Other Definitions

Here are the definitions for some critical security elements.

- Content Security Policy (CSP) - Defines which scripts and script sources an extension can interact with. Each Chrome extension must define its unique CSP in the manifest.json. By Default, same origin policies are applied (as of manifest version 2). Webpages and different domains generally have their own CSPs as well. The default CSP disables inline scripts and the use of Javascripts' *eval*; however, a user may choose to enable any of them. In case they are enabled (which is **not** recommended), chrome recommends to provide a base64 hash of the inline script [10, 11, 12, 13].
- Content Scripts - files that run in the context of web pages (can read web page details, make changes, and pass info to extensions). These sandboxes are essentially all powerful but isolated (not subject to the CSP of the extension or the web page itself).

4.3 Attack Vectors and Threat Model

Installing extensions onto your Chrome browser opens up a few avenues of attack that normally would not be available to malicious users. These can briefly be placed into three categories: malicious apps, malicious users on the network, and malicious users from the web.

Installing any software onto your computer carries with it risk. Google promises to remove any flagged apps but warrants that all those who download extensions onto their browser accept the inherent risks (see section 4.1). Extensions can request permissions to interact with your computer in ways that could be leveraged to cause damage, but the user is expected to acknowledge the risk at the time of installation. The way that extensions themselves interact with the internet and your computer can also cause negative repercussions.

Threat Model

We break down the threat to users into malicious extensions and extension vulnerabilities. Google Chrome's developer agreement [3] legally covers them against malicious extensions, and not much more can be done than existing policies such as the verification and report procedures Google currently offers. However, extension vulnerabilities are more subtle, and potentially far greater risks to most users. For example, a study on trusted password manager extensions including Lastpass found that most were susceptible to simple exploits [17]. Below we discuss the two main categories of extension vulnerabilities: (1) network attacks and (2) malicious website attacks.

Network Attacks

As extensions interact with the internet via API calls, a malicious user can piggyback off of the permissions and data of the extension to attack your computer. Network attackers can be either active and passive. An active attacker could corrupt an HTTP message and send code to

be executed by the extension or extract code from an extension through it's request, while a passive one may simply try to retrieve passwords or sign in tokens from your cookies.

Internet protocols have built-in protections against many of these attacks, but Chrome extensions have historically been vulnerable to two types of scripting attacks: Direct script evaluation via Same Origin Forgery, and scripts that aren't directly evaluated—HTTP scripts and HTTP Cross Domain Requests (XHRs) [18]. These attacks have been largely mitigated in recent years by the adoption of HTTPS. However they are also mitigated by extension specific and website CSPs when properly implemented.

Malicious Website Attacks

Extensions also interact with the web pages they run on, and can thus be exploited directly or indirectly via browser features. The most well known direct attack is an XSS attack, in which a malicious website abuses extension scripts interacting with the page by submitting a piece of javascript code into a textbox. When evaluated, that code can cause undesirable behaviour. Other attacks are much simpler. In the Lastpass password manager exploit, the extension was fooled into thinking it was on a trusted domain, so its autofill/auto-login functionality could be phished [17].

A similar scenario can occur with content hosted on web pages by malicious parties. This is the content that an extension interacts with, as an extension has certain permission and privileges, if these can be taken advantage of by a malicious user it could lead to new exploits and attacks on the user.

5. Results / Evaluation

Overall we managed to get information on a total of 2750 different extensions from the Chrome webstore. After running our software on these extensions and extracting the relevant pieces of data into a Google Sheets document (which can be found at this link: *docs.google.com/spreadsheets/d/1uYAADimCLRkkYQAlOSRclFuodI8Mi5iIQY61dezTWO0/edit?usp=sharing*) we found that more than half of all applications requested permissions for cookies, storage, access to open tabs, and context menus, with over 62% of all the examined apps requesting either storage, open tab access, or both (See Figure 5.1). All of these are particularly dangerous permissions that should be avoided when possible as they have the potential to reveal a user's private information and code to be put on a user's computer. It was found that the average extension requests 4.18 permissions and on average requests 2.4553 permissions that Chrome has identified as dangerous permissions. We must note that Chrome doesn't explicitly state that these permissions are "dangerous" but Chrome requires user approval of an extension that uses any of the permissions listed in Figure 5.2.

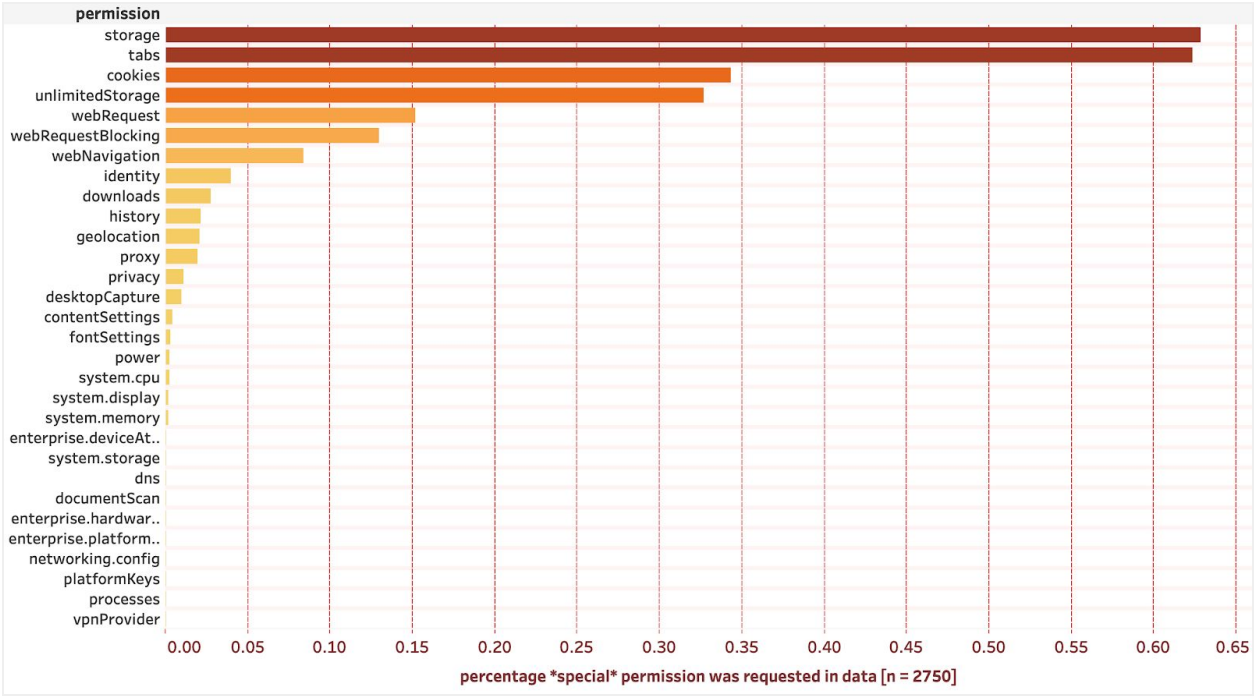


Figure 5.1: Percentage of special permissions requested in our dataset of 2750 extensions.

> *contentSettings, **cookies**, desktopCapture, dns, documentScan, **downloads**, enterprise.deviceAttributes, enterprise.hardwarePlatform, enterprise.platformKeys, fontSettings, geolocation, **history**, **identity**, networking.config, platformKeys, power, privacy, processes, proxy, storage, system.cpu, system.display, system.memory, **system.storage**, **tabs**, **unlimitedStorage**, vpnProvider, **webNavigation**, **webRequest**, **webRequestBlocking***

Figure 5.2: Special permissions—permissions that require user approval. We bolded the 10 permissions that were most requested among these.

Of particular concern are calls to the *eval* method, which directly take in an expression and runs it as code. These methods can easily be abused by malicious parties to inject code into a user's browser. This allows a malicious user to completely leverage an extension's permissions for control over the browser and whatever data may be there. There are multiple versions of *eval* that can be called with ranging levels of risk with the most hazardous simply running the code with no security checks. It was found that 914 out of the 2750 extensions we examined had instances of the *unsafe-eval* directive in their CSP, meaning that roughly a third of the apps chosen could be susceptible to code injection because for some reason they needed to allow the use of *eval*. Only 12 of the 914 extensions actually followed Chrome's directive to hash the part of the code in which they use *eval*, meaning that 902 extensions collected directly from the Chrome Web Store are susceptible to code injection.

Name	Is Good	Times Used	pct
manifest_version	1	2750	1
script-src:unsafe-eval	0	902	0.3280
script-src:unsafe-eval-base64-hash	1	12	0.0040
script-src:unsafe-inline	0	2	0.0017
script-src:unsafe-inline-base64-hash	0	2	0.0007
script-src:non-localhost-http-url	0	0	0.0000
style-src:unsafe-eval	0	4	0.0015
style-src:unsafe-eval-base64-hash	0	0	0.0000
style-src:unsafe-inline	0	48	0.0175
style-src:unsafe-inline-base64-hash	1	0	0.0000
style-src:non-localhost-http-url	0	5	0.0018
BAD-CSP	0	941	0.3422

Table 5.1.a: Aggregate results CSPs and the use of unsafe eval. In the above table, the *BAD-CSP* row is an aggregate count of how many extensions either have *script-src:unsafe-eval*, *script-src:unsafe-inline*, *script-src:non-localhost-http-url*, *style-src:unsafe-eval*, *style-src:unsafe-inline* or *style-src:non-localhost-http-url* set up. These extensions can easily be exploited.

Name	Is Good	Times Used	pct
default-src	1	55	0.0200
base-uri	1	5	0.0018
form-action	1	4	0.0015
frame-ancestors	1	7	0.0025
plugin-types	1	0	0.0000
report-uri	1	7	0.0025
sandbox	1	5	0.0018

Table 5.1.b: Aggregate results on CSPs on other recommended policies. Not many extensions use these. At best, only 55 (2%) of extensions use the *default-src* policy.

As discussed before CSPs are an important component for all security on the internet ensuring that only resources from trusted sources are loaded. Google offers developers the ability to set a CSP policy for their extension directly in the extension's manifest. It was found that 941of the 2750 extensions surveyed had used CSPs in ways that were directly opposed to what Chrome had recommended—that is, by using *unsafe-eval*, *unsafe-inline*, or allowing *http* sites in their CPSs for either the *script-src* or the *style-src* policies (see Google sheets for details).

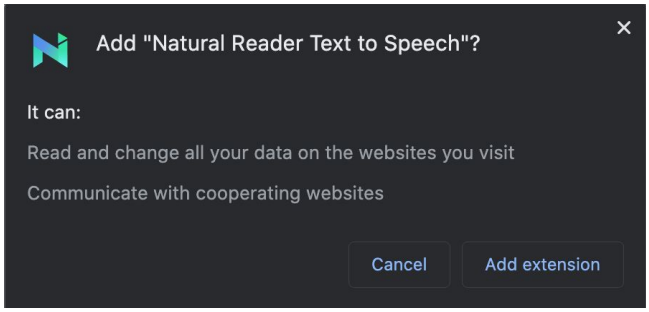
6. Discussion/Future work

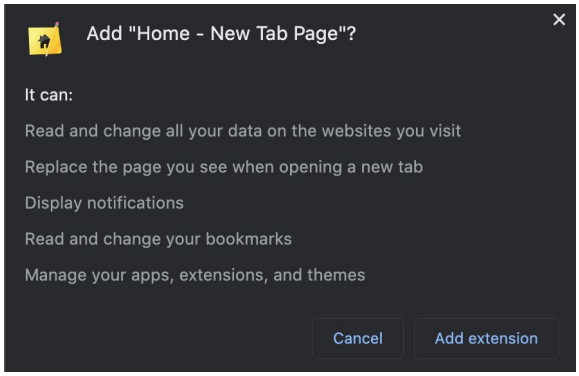
A lot of our work was focused on detecting known exploits and common vulnerabilities in Chrome extensions. Despite our analysis only focusing on a couple thousand apps, making up less than one percent of the webstores total contents, we were able to discover significant security concerns in many of the extensions we examined which may be abusable by malicious parties. The tool we developed provides a brief overview of these common exploits, but with enough work a far more advanced version could be fleshed out to generate far more detailed security reports on extensions.

The extension's permission model mostly just passes liability into the end user, as they actively agree to give the extensions access to what it requests, not Google. **While it does**

inform about some key components that an extension can interact with, it does not give enough information for the average user to understand how much risk they are taking on and it certainly does not solve the underlying security problems. It is a simple and indescriptive dialogue box that can actually be more misleading than helpful based on our criteria.

The following two extensions both have about 100,000 users, are published by seemingly reputable developers. Here are their respective warnings.





Although at a glance the “Home - New Tab Page” may seem more problematic, there is no indication that both of them have actually broken Google’s CSP security recommendations.

While the permissions framework can essentially be rendered useless with the click of a button, Chrome does not enforce the use of a good CSPs when creating an extension. With proper CSPs a lot of common exploits would be rendered useless. One alternative is to have the strictest CSP be the default, requiring developers to include a note explaining why they need a different CSP. This could either just be shown to the user so that they know the extension is lacking a secure CSP or even used by the Chrome team to randomly flag extensions for manual inspection.

There already exists 3rd party software developed by independent groups to grade extensions. One good example is the CRXcavator web app found at https://crxcavator.io/. This web app instantly generates a “Risk Score” for any given extension based on a myriad of properties. If Google were to use something to automatically filter extensions below a safety threshold, it could weed out a large portion of malignant extensions. This could work even better if all extensions had a publicly displayed security score on the webapp as well as a breakdown of why the extensions had that score. It would allow users to have a more informed perspective when it comes time to download an extension. Interestingly enough CRXcavator lists that “Home - New Tab Page” and “Natural Reader Text to Speech” both have critical CSP problems, and the same exact CSP Risk Score.

7. Conclusion

While Google has implemented an automated extensions verification process, little is known about how it actually operates and based on our results it seems that a lot of extensions still pose a significant security risk to their users. We suggest that Google overhaul their current

extension verification by not only making it stricter but providing a security score for any submitted extensions with a breakdown of possible security risks. This would be a simple addition that would not only help give users an idea of what risks they're exposing themselves to by downloading an extensions, but also let developers know what they could do to make their extension more secure. Although this would not resolve all the security threats from browser extensions, it would allow for more informed decisions and encourage safer extension development.

8. Citations

TODO: Reformat citations below.

[1] "User Agent Breakdowns." *Wikimedia Foundation*, analytics.wikimedia.org/dashboards/browsers.

[2] Chipman, Ryan, Tatsiana Ivonchyk, Danielle Man, and Morgan Voss. "Security Analysis of Chrome Extensions." 12 May 2016. courses.csail.mit.edu/6.857/2016/files/24.pdf

[3] "Google Chrome Web Store Developer Agreement." *Chrome*, Chrome, developer.chrome.com/webstore/terms.

[4] "Privacy Policy – Privacy & Terms." *Google*, Google, policies.google.com/privacy?hl=en-US.

[5] "Developer Program Policies." *Chrome*, Chrome, developer.chrome.com/webstore/program_policies.

[6] "Frequently Asked Questions." *Chrome*, Chrome, developer.chrome.com/webstore/faq.

[7] "Browser Market Share Worldwide." *StatCounter Global Stats*, gs.statcounter.com/browser-market-share#monthly-201910-201910-bar.

[8] "Browser Market Share." *Browser Market Share*, NetApplications, netmarketshare.com/browser-market-share.aspx.

[9] Cimpanu, Catalin. "Half of All Google Chrome Extensions Have Fewer than 16 Installs." *ZDNet*, ZDNet, 3 Aug. 2019, zdnet.com/article/half-of-all-google-chrome-extensions-have-fewer-than-16-installs/.

[10] "Content Security Policy (CSP)." Chrome, Chrome, developer.chrome.com/extensions/contentSecurityPolicy.

[11] "Manifest Version." *Chrome*, Chrome, developer.chrome.com/extensions/manifestVersion.

[12] "Content Security Policy Level 3." *World Wide Web Consortium (W3C)*, World Wide Web Consortium (W3C), 28 Feb. 2019, w3c.github.io/webappsec-csp/.

[13] West, Mike. "An Introduction to Content Security Policy - HTML5 Rocks." *HTML5 Rocks - A Resource for Open Web HTML5 Developers*, 15 June 2012, html5rocks.com/en/tutorials/security/content-security-policy.

[14] Chen, Benjamin, Tony Ding, Xiaolu Guo, and Adelaide Oh. “Exploiting Leaky Chrome Extension API.” 15 May 2019 https://courses.csail.mit.edu/6.857/2019/project/10-Oh-Chen-Ding-Guo.pdf

[15] Hay Newman, Lily. “Google Is Finally Making Chrome Extensions More Secure”. 30 May 2019. https://www.wired.com/story/google-chrome-extensions-security-changes/

[16] Gartenberg, Chaim. “Google announces changes to Chrome Web Store policies to help fight spammy extensions” 30 Apr 2020 https://www.theverge.com/2020/4/30/21242597/google-chrome-web-store-new-spam-policy-extensions

[17] Carr, Michael, and Siamak F. Shahandashti. "Revisiting Security Vulnerabilities in Commercial Password Managers." arXiv preprint arXiv:2003.01985 (2020).

[18] Fernandes, Earlence, et al. "Flowfence: Practical data protection for emerging iot application frameworks." *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 2016.

MISCELLANEOUS

Analysis of the Current Extension verification Process

- Possible loopholes in guideline/process?

Writing Test Extension and Exploiting Permissions (> sort of deferring this)

The chrome extension system operates in a model where the user is responsible

- Clickjacking
- Exploits that don't require permissions
    - (robertv) I can write a little bit on the permissions I wrote about
- Unintended effects of the permission system

Experimentation with existing extensions (> sort of deferring this)

- Can the data stored by other extensions be accessed
- Can you leverage an installed extensions permissions or remove them from another extension